INFORME TÉCNICO

Resumen de la aplicación Desarrollada:

Se basa en un sistema de Reservas de un restaurante con lógica de usuarios implementada.

En resumen, consiste en permitirle a un Cliente realizar, modificar, ver, y eliminar sus reservas, además de modificar su email y/o clave de usuario.

Mientras que el administrador implementa todas las funcionalidades que ofrece el restaurante (Platos, Mesas, obtener las Reservas).

Se verá más detallado en el Manual de Usuario.

Como proceso de desarrollo del sistema implementamos cada clase con su gestor, y luego desarrollamos la funcionalidad para la persistencia.

Probamos cada una de las funcionalidades antes de proceder con el menú visual.

Informe Técnico de la aplicación:

El sistema se basa en un ABM (Alta-Baja-Modificación) utilizando una lógica de usuarios.

Por lo tanto, tenemos:

Login(Diferente para Clientes y Administradores) con sus respectivas funcionalidades en cada caso.

El sistema permite:

Gestionar Reservas (Propias de cada cliente, el administrador las puede ver únicamente, las puede manipular manualmente por medio de la **"base de datos"** en caso de ser necesario).

Gestionar Mesas (Administrador).

Gestionar Platos (Administrador).

Gestionar Usuarios (Administrador únicamente bajas y vistas, el cliente se puede registrar).

Los datos del sistema los persistimos en archivos JSON basándonos lo más posible en la estructura de una base de datos de tipo Relacional.

Para ello utilizamos una clase GestorJSON para implementar las funcionalidades.

Archivos Base de Datos:

- reservas.json
- users.json
- mesas.json
- menu.json

Clases utilizadas, junto a su gestor:

Usuario (Cliente y Administrador como extensiones) ---------- GestorUsuarios

Mesa ---------- GestorMesa

Reserva ---------- GestorReserva

MenuItem (Plato) ---------- GestorMenuItem

TipoPlato (En enum para definir el tipo de plato (entrada, plato principal, postre))

Sistema (Clase que ejecutará el sistema con un método run())

GestorJSON (Funcionalidades de la base datos (En JSON, por que se hacían los importantes =P))

ManejadorMenu (Un menú visual implementando la funcionalidad desarrollada)

Decisiones tomadas en el diseño:

Se trataron de llevar a cabo las reglas de los principios **S.O.L.I.D.**

Para lo cual, diseñamos las clases principales junto a un Handler (gestor) que lleva a cabo la funcionalidad. (Principio de Responsabilidad Única).

Los usuarios tratamos de basarlos en el Principio de Sustitución de Liskov (además para implementar herencia).

Para los demás principios, tales como el Principio de Abierto/Cerrado y Principio de Inversión de Dependencias: **Favorecimos la composición sobre la herencia, para tener un código más mantenible y generar islas de cambio en lugar de mucho acoplamiento.**

En el caso del Principio de Segregación de interfaces, no lo aplicamos porque no utilizamos interfaces (¿Se viene en el Sprint 2? No lo sabremos... ¿O sí??).

Plan de Trabajo (Matriz de soluciones y diario de trabajo):

Utilizamos metodología Ágil, basado en un Sprint de 2 semanas para la primera demo (y única considerando el alcance del TP).

Con reuniones diarias (dailies) de 25 minutos aproximadamente, para conversar las tareas de cada miembro y despejar dudas.

En cuanto al **control de versiones**, utilizamos **GIT** con separación en ramas para que cada miembro del equipo pueda desarrollar sus tareas sin ser bloqueado ni bloquear al resto en la medida de lo posible.

Link: REPOSITORIO GIT DEL PROYECTO

Un miembro del equipo se encargaba de realizar el merge con la rama master una vez que se agregaba funcionalidad.

Para la **división de tareas** nos basamos en un sistema Kanban de tickets utilizando la plataforma **Trello**.

Link: BOARD TRELLO

Conclusiones:

Como una conclusión general, logramos finalizar a tiempo y no tuvimos inconvenientes (Salvo en la bendita reserva) ya que aplicamos un flujo de trabajo organizado.